**Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference**
**IDETC/CIE 2008**
**August 3-6, 2008, Brooklyn, New York, USA**

**DETC2008-50137**

# GENETIC PROGRAMMING OF AN ARTIFICIAL NEURAL NETWORK FOR ROBUST CONTROL OF A 2-D PATH FOLLOWING ROBOT

**Anthony M. Roy**
Engineering Design and Research Laboratory
California Institute of Technology
1200 California Blvd.
Pasadena CA 91125
email: roy@design.caltech.edu

**Erik K. Antonsson**
Engineering Design and Research Laboratory
California Institute of Technology
1200 California Blvd.
Pasadena CA 91125
email: erik@design.caltech.edu

**Andrew A. Shapiro**
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena CA 91109
email: aashapiro@jpl.nasa.gov

## ABSTRACT

*Genetic Programs that have phenotypes created by the application of genotypes comprising rules are robust and highly scalable. Such encodings are useful for complex applications such as controller design. This paper outlines an evolutionary algorithm capable of creating a controller for 2 DOF, path following robot. The controllers are embodied by Artificial Neural Networks capable of full functionality despite multiple failures.*

## 1 Introduction

Encoding schemes that apply embryogenesis are emerging as a new way to encode genotypes for Genetic Algorithms (GA). Theraulaz [1] and Pollack [2] have shown that the reuse of a small set of rules to create a phenotype is an effective alternative to storing and manipulating the large amount of data that describes each individual directly. Bently and Kumar [3] have shown that indirect encodings produce solutions to design problems faster and better than their directly encoded counterparts. Federici and Downing [4] have shown that rule-based encoded designs are more robust as well. Furthermore, these encodings are much more likely to create modular designs, another desirable quality for many designs [5].

Considering these many advantages, indirect encodings have been used for a myriad of design problems, among them creating artificial neural nets (ANNs) for robotic control [6], [7], [8], [9]. The complexity and size of ANNs make them attractive candidates for Evolutionary Computation (EC) applications. However, many of these schemes use direct encodings, which lack many of the aforementioned advantages of developmental methods. Futhermore, well-established GAs that do use embryogenesis feature tree-braching developments [10] which are difficult to decypher and are not readily reconfigured by natural genetic operations as they exist in nautre. Grammatical Evolution (GE) has been recently created to adress the problem of programming in any langauge [11]. While most of the work done in GE has been purey in the realm of computer science, there have been some applications of GE for ANN creation [12]. However, these methods do not address concerns for robotic manipulation.

This paper explores a GE encoding scheme for the creation of ANNs to use as a controller of a line-following robot. The genome is a set of variable-length rules that are decoded to create a $C^{++}$ program. The $C^{++}$ programs used to create the ANNs have an *IF then ACTION* structure. Each program has multiple sections that cycle through each node with tests and actions of the form:

> *IF Node* $\alpha$ *and/or Node* $\beta$ *meets certain criteria, perform ACTION.*

As with Evolutionary Computation itself, the work presented in this paper attempts to mimic nature in a manner much closer than previous methods. Rather than using a tree like structure as in most ANN embryogenesis, the genome is represented as a series of numbers ranging between 1 and 4. This representation allows

one to use mutations such as point substitution, crossover, and gene duplication/deletion in a manner much closer to their biological counterparts. Furthermore, an intermediate step is the creation of $C^{++}$ programs which create the actual protein. Having these $C^{++}$ programs is beneficial in seeing how the ANN was created as anyone able to understand $C^{++}$ is able to read the rules used to create the ANN. While the test subject is a small, feed forward ANN controller, the nature of rule-based encodings allows them to construct much larger ANNs.

## 2 Description of Individuals

The goal is to evolve a controller for a robot, even as multiple nodes and connection are removed from the ANN. Next is a review of the various components of each individual. Each individual is comprised of a simulated two-wheeled robot equipped with photovoltaic sensors. Furthermore, the controller of each robot will be a network of McCulloch-Pitts neurons. Each ANN is created by the execution of the rules encoded in its genome. When the genomes are decoded, the result is a $C^{++}$ script. When the script is compiled and executed, the aforementioned ANN is created.

### 2.1 Phenotype - Path Following Robot

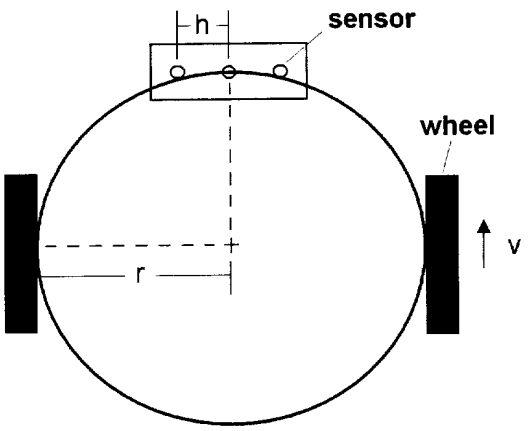



Figure 1. Model of Path-Following Robot

The robot model is shown in figure 1. Each wheel is configured to provide a set linear speed, v. The three photovoltaic sensors are set up so that the sensor is *on* when it is over a dark section, and *off* when over a light one. Both the robot's position and sensor status are updated in increments of $\Delta t = 0.02$ seconds. The path to be followed is a line with a width of w and a centerline satifies equation 1. The constants are chosen to ensure the line intersects the origin with a slope less than $\frac{h}{r}$. Furthermore, the curvature of the line is always less than the turning radius of the robot, $\frac{1}{r}$. The each sensor is configured to *on* when over the line and *off* otherwise.

$$y(x) = A\cos(fx) + Cx - A \tag{1}$$

### 2.2 Phenotype - ANN Controller

The sensors serve as the inputs to the ANN, while each output of the ANN controls a wheel. Each ANN is composed of McCulloch-Pitts modeled neurons shown in Figure 2. The neuron sums the weighted inputs, $w_i$, then inputs the sum into a Heaviside function with a threshold of $t$, as shown in equations 2 and 3. The ANN is updated at the same time as the robot's position, so large ANNs can experience noticable lag times. The node's output, $O(u)$, maybe weighted before it is used as an input for another node. However, $O(u)$ for an output node is always unweighted, resulting in binary outputs for the entire ANN.

$$u = \sum_{i=1}^{n} w_i \tag{2}$$

$$O(u) = \begin{cases} 1 \text{ if } u > t \\ 0 \text{ if } u \leq t \end{cases} \tag{3}$$

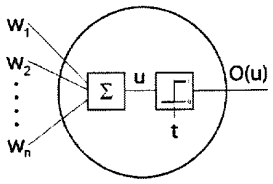


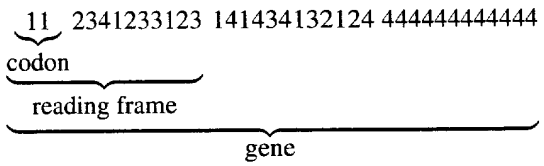

Figure 2. McCulloch-Pitts Neuron Model

Finally, each node is denoted by two different types. Type 1 values are letters which range between A - F, I, and O. While A - F can be any hidden node, I and O are reserved for input and output nodes, respectively. Type 2 values are numerical values that range between 1 and 8, and nodes are enumerated in the order in which they are created. Thus, the third B node made by the ANN would have a *type 1* of B and a *type 2* of 3. If more than 8 nodes of a type 1 are created, the enumeration begins again. Thus, the ninth node with a type A would be labeled A1.

## 2.3 Biological Analog

It would be helpful to review the biological analogy which was the inspiration for this particular encoding scheme. The genome of each individual is an array of integers which is decoded to create C$^{++}$ programs. Similar to the quaternary system of natural genetics, every digit is a *nucleotide* whose value is $n = \{\mathbb{Z}|[1,4]\}$. It takes a pair of nucleotides to write anything into the C$^{++}$ script. Thus two nucleotides are analogous to a *codon* transcribing an amino acid into a protein. A collection of six codons (twelve nucleotides) forms a complete if/then statement, which can be thought of as the secondary structure of a protein. However, these tests are not independent, and the sequence of the tests will greatly influence how the individual will grow. Therefore, a combination of these tests determines what actions will be performed and can be thought of the overall protein. Herein, a *reading frame* is a collection of six codons and the sequence of frames that creates an entire protein as a *gene*.

Figure 3. Sample Genome and Biological Analog




## 2.4 Genotypes and Genetic Decoding

The first pair of nucleotides in a frame determine the logical structure of the entire protein as governed by Table 1.

*if* - Opens an if statement. Adds action to the action stack
*end-if* - Executes action stack. Removes most recent action from action stack. Closes an if statement. Opens another if statement. Adds action to the action stack
*end-end-if* - Executes action stack. Removes most recent action from action stack. Closes an if statement. Executes new action stack. Removes action from action stack. Closes an if statement. Opens another if statement. Adds action to the action stack
*end-del* - Executes action stack. Closes an if statement.
*end-end-del* - Executes action stack. Removes most recent action from action stack. Closes an if statement. Executes new action stack. Removes action from action stack. Closes an if statement.
*end-ALL* - Executes action stack. Removes most recent action from action stack. Closes an if statement. Repeats until all if statement are closed.

The second codon dictates what attributes will be tested. The attributes are: a node's first or second type; a node's threshold;

Table 1. 1st Codon - If Structure

| | | $N_2$ | | | |
|---|---|---|---|---|---|
| $N_1$ | | 1 | 2 | 3 | 4 |
| | 1 | if | if | if | end if |
| | 2 | if | end if | end if | end *del* |
| | 3 | if | end end if | end end if | end end *del* |
| | 4 | end end if | end *del* | end end *del* | end ALL *del* |

$N_1$ = NUCLEOTIDE 1 $N_2$ = NUCLEOTIDE 2

Figure 4. If Structure Codon and Protein Transcription

```
                                    if (a)
                                        if(b)
if 11(Test a)(Action A)                     B
                                            A
if 12(Test b)(Action B)                 end
                                        A
                                    end
end-end-if 32(Test c)(Action C)     if (c)
                                        if (d)
if 13(Test d)(Action D)                     D
                                            C
end-del 42(Test e)(Action E)            end
                                        if (f)
if 21(Test f)(Action F)                     F
                                            C
end-ALL 44(Test g)(Action G)            end
                                        C
                                    end
```

the quantity of outputs from a node; or the number of inputs into a node. Furthermore, each attribute tested is that of a pair of nodes labeled Node $\alpha$ or Node $\beta$. The program tests all pairing permutations of nodes. The third codon determines what mathematical operator the test will use. The operators are equal, not equal, greater than, and less than. The fourth pair of nucleotides assigns the values the attribute are tested against. Type values range between the letters A - F, I, and O. While A - F can be any hidden node, I and O are reserved for input and output nodes respectively. Thresholds range between $\pm 1.75$ in discrete intervals of 0.25. Weights range between $\pm 1$, also at discrete intervals of 0.25. The number of inputs and outputs can range between 0 and 15. For all values, a Grey's type system is encoded such that all sequential values are a single bit flip away.

The fifth codon determines what action will be placed into the action stack. As aforementioned, a "stack" of actions is written into the program whenever an if statement is closed. Some of the permutations will result in the creation of a new node. Others will create a connection between Node $\alpha$ and Node $\beta$. In both these cases, the last two nucleotides dictate the threshold of the new node or weight of the connection, respectively. The transcription options are identical to the bias and connection values of the fourth codon. However, there are also several *No Action* blocks which will not insert any new action commands.

### 2.5 C$^{++}$ Programs (Proteins)

Each C$^{++}$ program is a collection of proteins that build the phenotype. While the genome creates the bulk of the algorithm, there are a few rules hard coded into the C$^{++}$ script of every individual. First, every ANN begins as two input nodes with a threshold of zero. As there is no option to create another input, each ANN will contain exactly two input nodes. Furthermore, the two inputs are unable to connect to each other . Also, each input, and all subsequent nodes thereafter, can create up to one additional node of either a hidden layer type or output type.

As we are only considering feed-forward ANNs, nodes are only able to make connections to nodes created after them. Furthermore, the creation of an output node will stop the creation of any other nodes. However, connections may still be grown at this point. The act of creating a node or connection consumes one of the individual's predetermined energy units for the entire ANN. The individual is considered to be completely developed once the individual uses all energy units or the programs cycles through all pairing permutations without performing any actions. Thus, to create larger ANNs, one only has to allot more energy units. These hard coded rules are implemented to impose the minimum constraints any viable feed forward ANN would have, while leaving enough flexibility to create a variety of architectures.

## 3 Genetic Algorithm

Each evolutionary run begins with the random creation of 192 individuals, each having genome lengths of 500 nucleotides. After the embryogenesis of each ANN, as described by the method above, each individual is evaluated. The fitness function uses a modified tier system with an individual being rewarded exponentially for each goal. Equation 4 is the fitness function used for evaluating individuals. The exponent in Eq 4 is decided by the tier system in Table 2 . The first tier ensures the individual grows the correct number of output nodes. In the second tier, the exponent is increased as each output is connected. These two requirements are the minimum for any possibly viable controller, and once met, will yield an exponential value of $x-1=1$. The robot simulation begins and the robot is allowed to runs for 20 seconds. After the 20 second simulation, the robot's path is evaluated by equation 5 where $r$ is the robot's radius, $w$ is the thinkness of the path, and $\varepsilon$ is the root mean square of the error between the robot's center and the centerline of the path. This value is added to the exponent. $f(\varepsilon)$ values greater than .75 allow the individual to prgress to tier 4.

$$Fitness = \lfloor 4^{x-1} \rfloor \tag{4}$$

$$f(\varepsilon) = 1 - \frac{1}{1+exp(-4(\frac{\varepsilon - r}{w}))} \tag{5}$$

In tier 4, a connection is randomly broken and the robot is simulated again. Connections are continually broken until the robot's path-following abilities fall below the tier 4 threshold. Once the circuit fails, the connections are replaced and the process is repeated with broken nodes. This test for robustness is done once for each generation the individual is alive. Because the order in which the connections and nodes are removed changes each generation, an individual's fitness is not constant, and its overall robustness is repeatedly tested.

Table 2. Tier for adjusting fitness exponent ($x$)

| Tier | Test | Change in Exponent |
|---|---|---|
| 1 | Are there enough output nodes? | % of desired output nodes |
| 2 | Are there a connections to each output node? | + % of output nodes with connections |
| 3 | Simulate robot for 20 seconds | + % of path followed correctly |
| 4 | Break connections and nodes until failure | + % of connections broken + % of nodes broken |

A roulette style of selection determines which individuals are used for creating the next generation, and population size is conserved. The probability of selecting any one individual is its fitness value divided by the summed fitness of the entire population. 25% of the population of the current generation live on to the next generation. Another 25% are created by making random interger switches of sigular parents. Each locus has a 7.5%

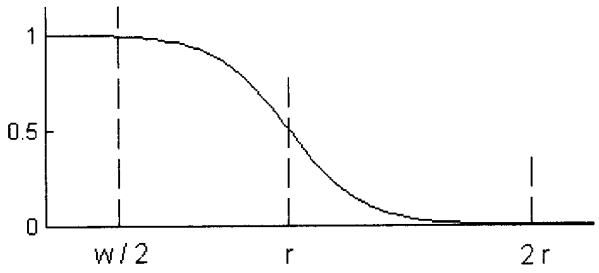



Figure 5. Preference Function for Position Error

chance of being replaced by a different nucleotide. Yet another 25% are created using single-point crossover, with two parents creating two children. The final 25% are created by gene duplication/deletion. This mutation takes a random section of the genome and either copies it again or deletes it entirely. There is an equal probability of either event happening to account for the biological phenomena of deletion. Finally, the same individuals aren't used for every process, so while an individual may not survive, it may still pass on much of its genetic information through another means of selection. This genetic algorithm is iterated for 200 generations.

## 4 Experimental Results

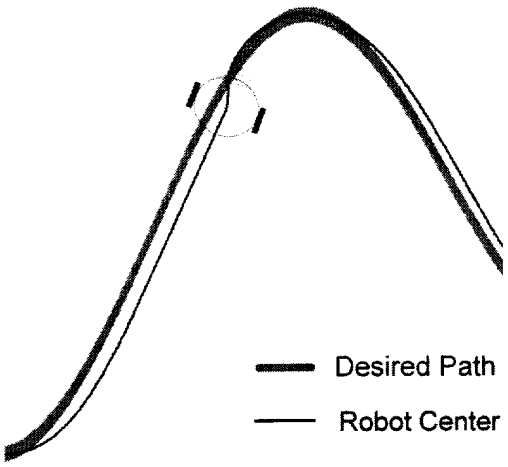



Figure 6. Robot Path verses Desired Path

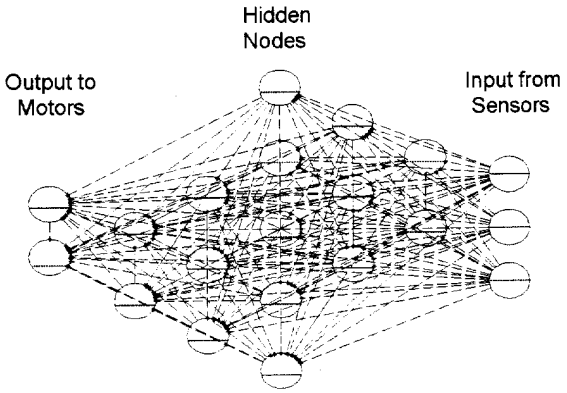



Figure 7. ANN Controller for Path Following Robot

Figure 6 shows the desired path overlayed by the path taken by the robot's center. The ANN controller of the robot is shown in figure 7. In this representation, the threshold of a node is depicted by the height of a horizontal chord in each node, with a threshold of 0 being the diameter. A solid connection indicates a positive weight while a dashed connection is indicative of a negative weight. The thickness of the line is proportional to the magnitude of the weight it represents.

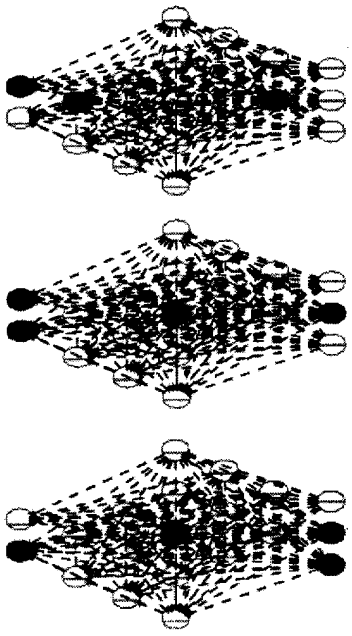


Figure 8. Functionality of ANN Controller

This is a rather large, confusing network, so some aspects of its functionality is shown in figure 8. The controller commands the robot by turning left when no sensors are active, forward when only the middle sensor is active, and right when the middle and right sensor are active. While one may expect the robot to turn left whenever the left sensor is activated, this actually result it a constant error for the robot's center due to the distance between the center and sensors. With the above configuration, the robot overshoots left turns, but is able to eliminate erros on right turns.

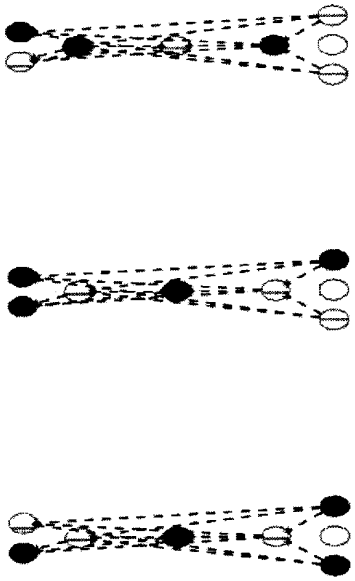


Figure 9. Functionality of ANN Controller Several Nodes Damaged

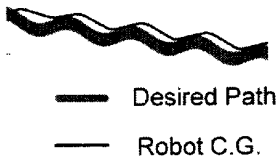



Figure 10. Robot Path with a Sensor and ANN Dammaged

Figure 9 shows the new functionality of the controller with 80% of the hidden nodes removed and the middle sensor broken. Figure 10 shows the new line following abilities of the robot with the broken controller. Although there is now a constant error, the robot still follows the overall sinusoid dictated by the desired path.

## 5 Conclusion

Overall, the GA presented herein was able to produce a robust controller for a line following robot within a limited evolutionary search. One can see that the control logic produced in figure 8, while unintuitive for a human designer, keeps the center of the robot close to the desired path. Furthermore, the controller is able to compensate for failures via node and connection removal. The ability to created an ANN controller that is able to function at an %80 failure rate and the loss of a sensor is a difficult design task at best. The authors believe that the flexibility of this GP is integral in having an overall GA capable of efficiently exploring a discontinuous fitness landscape while simultaneously converging to good answers. Future work includes the extension to more complicated contorllers, particularly ones with feedback. Furthermore, implementing learning schemes, one of the core abilities of biological neural networks, will be integral to the development of any GA/ANN scheme, including this one.

## 6 Acknowledgments

Part of the research described in this paper was sponsored by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## REFERENCES

[1] Pollack, J. B., Hornby, G. S., Lipson, H., and Funes, P. Computer creativity in the automatic design of robots.

[2] Theraulaz, G., and Bonabeau, E., 1995. "Coordination in distributed building". *Science,* **269**(5224), August, pp. 686–688.

[3] Bentley, P., and Kumar, S., 1999. "Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem". In Genetic and Evolutionary Computation Conference, pp. 35–43.

[4] Federici, D., and Downing, K., 2006. "Evolution and development of a multicellular organism: Scalability, resilience, and neutral complexification". *Artif. Life,* **12**(3), pp. 381–409.

[5] Koehn, P., 1996. Genetic encoding strategies for neural networks.

[6] Lewis, M., Fagg, A., and Bekey, G. "Genetic algorithms for gait synthesis in a hexapod robot".

[7] Lewis, F. L., 1996. "Neural network control of robot manipulators". *IEEE Expert: Intelligent Systems and Their Applications,* **11**(3), pp. 64–75.

[8] Floreano, D., Mitri, S., Magnenat, S., and Keller, L., 2007. "Evolutionary conditions for the emergence of communication in robots.". *Curr Biol,* **17**(6), Mar, pp. 514–519.

[9] Floreano, D., Drr, P., and Mattiussi, C., 2008. "Neuroevolution: from architectures to learning". *Evolutionary Intelligence.*

[10] Gruau, F., 1993. "Genetic synthesis of modular neural networks". In Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., pp. 318–325.

[11] O'Neill, M., and Ryan, C., 2001. "Grammatical evolution". *IEEE Trans. Evolutionary Computation,* **5**, pp. 349–358.

[12] Tsoulos, I. G., Gavrili, D., and Glavas, E., 2005. "Neural network construction using grammatical evolution". *IEEE International Symposium on Signal Processing and Information Technology*, pp. 827–831.

[13] Astor, J. C., and Adami, C., 2000. "A developmental model for the evolution of artificial neural networks". *Artificial Life,* **6**(3), pp. 189–218.

[14] Stanley, K. O., and Miikkulainen, R., 2002. "Evolving neural networks through augmenting topologies". *Evolutionary Computation,* **10**(2), pp. 99–127.

[15] Kitano, H., 1990. "Designing neural networks using genetic algorithms". *Complex Systems,* **4**(4), pp. 461–476.

[16] Ohno, S., 1970. *Evolution by gene duplication.* Springer-Verlag.

[17] Ohno, S., Wolf, U., and Atkin, N. B., 1968. "Evolution from fish to mammals by gene duplication". *Hereditas,* **59**, pp. 169–187.

[18] Lipson, H., and Pollack, J. B. "Automated design and munfacture of robotic lifeforms".

[19] Kartalopoulos, S. V., 1996. *Understanding Neural Networks and Fuzzy Logic.* IEEE Press, Piscataway, NJ.